Intermediate Programming in R using Tidyverse

Getting Started

Within the R community, we observe that there are multiple code styles. Your code style will not matter as much if you are the only person analyzing the data, however, this becomes problematic if you work as a team along with other R programmers. To circumnavigate this issue, I will introduce a collection of packages (and philosophy) known as Tidyverse (https://www.tidyverse.org) and its code-style developed by Tidyverse and Google. Furthermore, we will look at some of the ggplot functions to visualize your data.

Table of Contents

1. Tidyverse Code Style Guide

1a. Naming R Script

File names should be meaningful, descriptive, yet concise. Avoid using special characters in file names and stick with letters, numbers, `-`, and `_`.

```
# good
fit_models.R
run_GWAS_analyses.R

# bad
analysis1.R
foo.R
```

If files should be run in a particular order, prefix them with numbers.

```
00_download_data.R
01_preprocess_data.R
02_analyze_data.R
03_plot_figures.R
```

1b. Organizing Script Structure

Also, it is a good idea to use commented lines of `-` to break up your file to readable chunks. Also, it is a good practice to load all of the libraries at the beginning of your scripts rather than sprinkling them throughout the code.

```
# Loading the packages -------------------------
library(package1)
library(package2)

# Loading the data -----------------------------
codes_to_load_the_data()

# Modifying the loaded data --------------------
modifying_the_loded_data()
```

1c. Syntax

Every language has a different syntax style. Here, I will show you the Google's (which is Tidyverse's) syntax.

Variable and **function** names should use only **lowercase letters, numbers, and** `_` . Here, you might notice that all of the function names include `_` whereas some of the base `R` codes use `.` . Long story short, you should reserve `.` for S3 object system.

```
# Good
loaded_data
modified_data
final_data
preprocessing_data()

# Bad
loadedData
LoadedData
data1
modifiedfinaldata
func1()
preprocessing.data()
```

Generally variable names should be nouns and function names should be verbs.

Spacing

Always put a space after a comma just like in regular English.

```
# Good
data_frame[, 1]

# Bad
data_frame[,1]
data_frame[ ,1]
data_frame[ , 1]
```

Parentheses

Do not put spaces inside or outside parentheses for regular function calls.

```
# Good
mean(values, na.rm = TRUE)

# Bad
mean (values, na.rm = TRUE)
mean( values, na.rm = TRUE)
```

Place a space before and after `()` if it is used for `if`, `for`, or `while`.

```
# Good
if (debug) {
  show(messages)
}

# Bad
if(value){
  show(messages)
}
```

Extra Spaces

Adding extra spaces is okay if it improves alignment of `=` or `<-`

```
value          <- 10
modified_value <- value + 20
```

Long lines

If a functional call is too long to fit on a single line, use one line each for the function name, each argument, and the closing.

```
# Good
do_something_very_long(something = "that", requires = many, arguemnts = "some of which may be lo
ng")

# Better
do_something_very_long(
  something = "that",
  requires = many,
  arguemnts = "some of which may be long"
)

# Bad
do_something_very_long("that", requires, many, arguments,
                       "which can be very long")
```

Assignment

One thing that is very different for R is that you have to use `<-` instead of `=` for assignment

```
# Good
x <- 5

# Bad
x = 5
```

1d. Pipes

This is one of the reason on what makes `Tidyverse` really unique! It uses `%>%` to emphasize a sequence of actions, rather than the object that the actions are being performed on.

You should use the pipe when:

1. You have multiple functions being done for a variable

You should avoid using the pipe when:

1. You need to manipulate more than one object at a time.
2. There are meaningful intermediate objects that could be given informative names.

```
# Traditional R
result <- function1(function2(function3(function4(x))))

# or ....
processed_data    <- process_data(initial_data)
modified_data     <- modify_data(processed_data)
filtered_data     <- filter_data(modified_data)
result            <- produce_result(filtered_data)

# By running your code this way, you are creating unnecessary variables that will take up your R
environment.
# Since R is heavily dependant on your RAM, more variables you have = slower the computation.

# Tidyverse R (Pipes)
result <- initial_data %>%
  process_data() %>%
  modify_data() %>%
  filter_data() %>%
  produce_result()
```

Instead of writing hard-to-read code and creating multiple unnecessary variables, you can utilize the pipes!

Long and Short Pipes

If the arguments to a function don't all fit on one line, put each argument on its own line and indent.

A one-step pipe can stay on one line. It is better to write it to a regular function call.

```
process_data(initial_data)
```

Whitespace

`%>%` should always have a space before it and should usually be followed by a new line as shown above.

Assignment

There are three acceptable forms of assignment

```
# Good
result <- initial_data %>%
  process_data() %>%
  modify_data() %>%
  filter_data() %>%
  produce_result()

result <-
  initial_data %>%
  process_data() %>%
  modify_data() %>%
  filter_data() %>%
  produce_result()

# Okay
initial_data %>%
  process_data() %>%
  modify_data() %>%
  filter_data() %>%
  produce_result() ->
  result
```

Pick whatever makes more sense or looks good to you.

Additional Resources

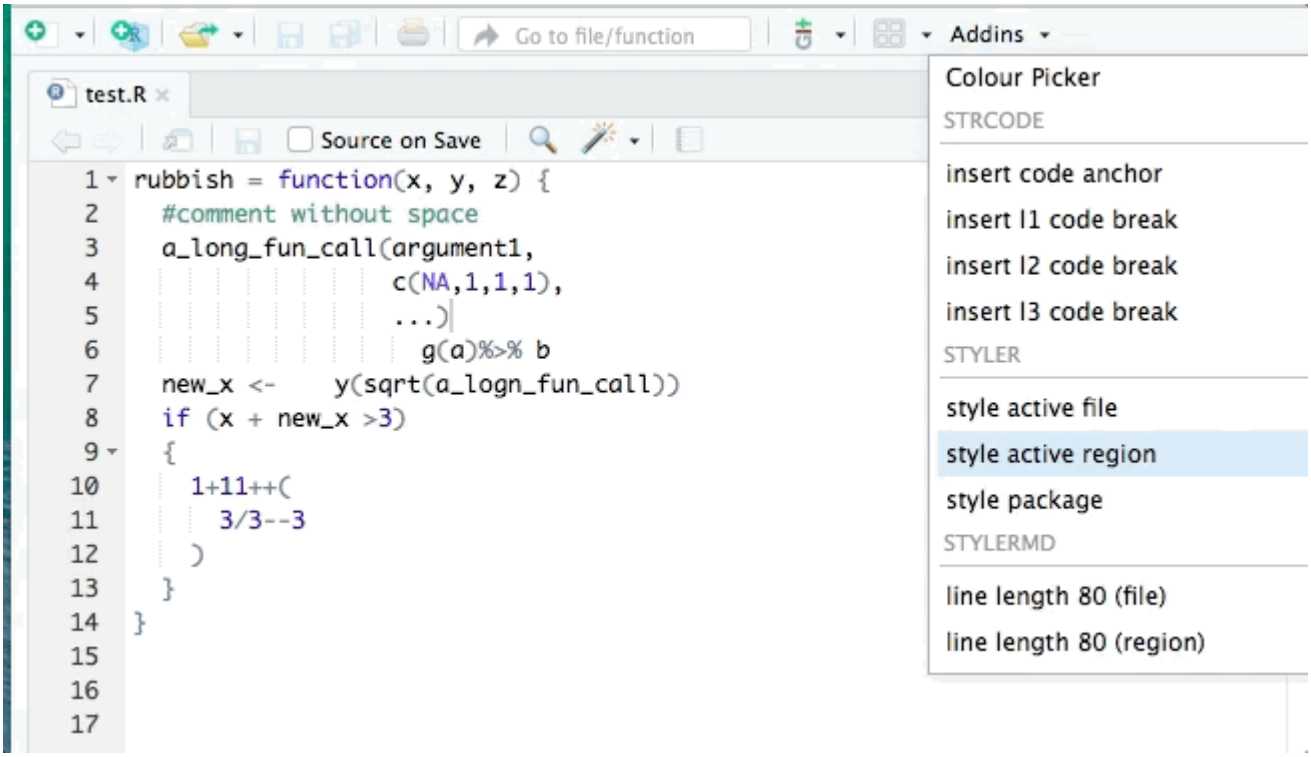
Pipes Manual (https://magrittr.tidyverse.org/reference/pipe.html)

Aliases Manual (https://magrittr.tidyverse.org/reference/aliases.html)

Tidyverse Packages (https://www.tidyverse.org/packages/)

Tidyverse Style Guide (https://style.tidyverse.org/files.html)

After writing all of your code, you can simply check your syntax by using `styler` package.



You can download this package by using either

```
# this
install.packages("styler")

# or this
remotes::install_github("r-lib/styler")
```

Examples

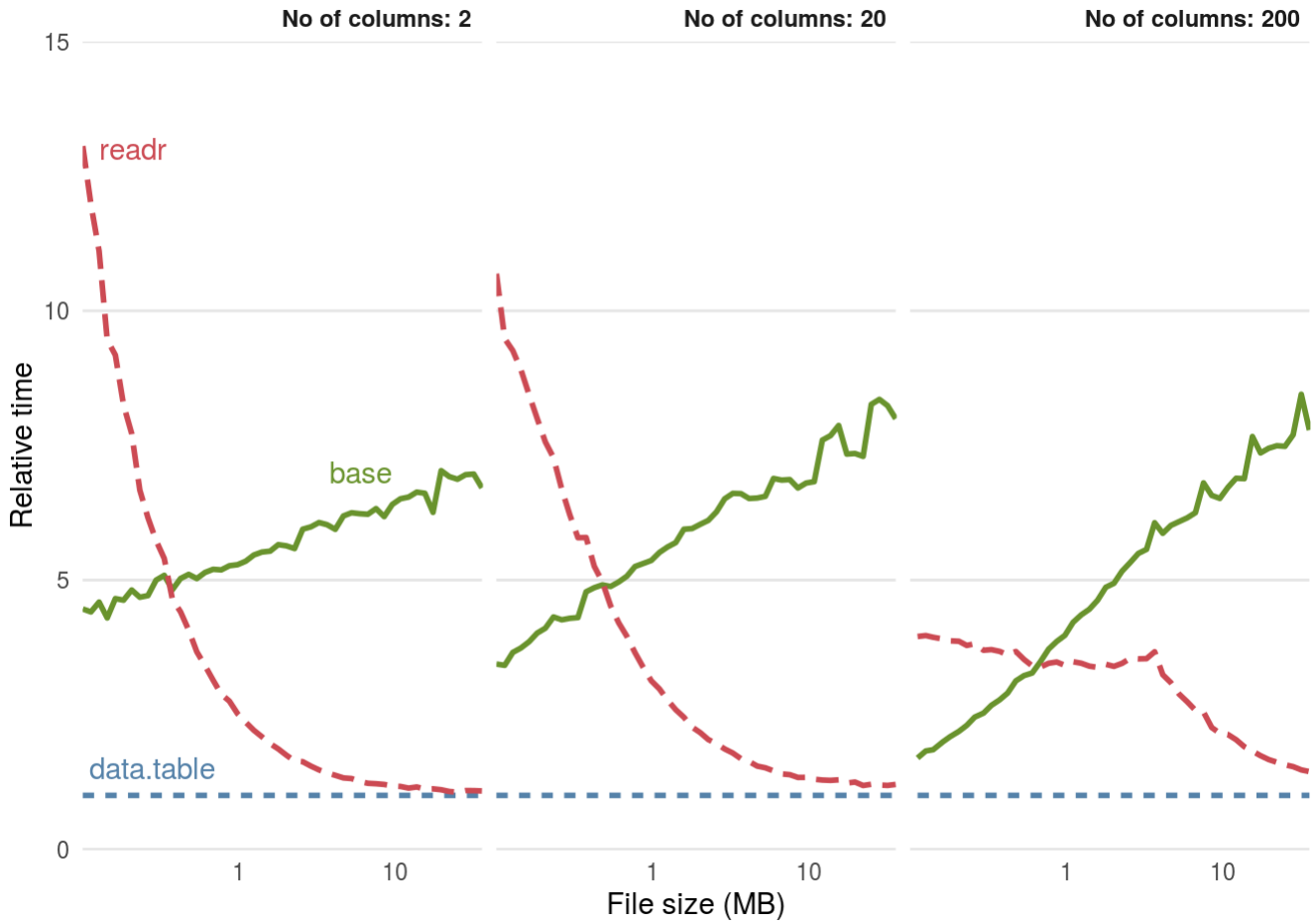
Let's look at a large environmental chemistry dataset. This example dataset was generated through chemical speciation analysis of smoke samples collected during lab-based simulations of wildfire events. Specifically, different biomass materials (eucalyptus, peat, pine, pine needles, and red oak) were burned under two combustion conditions of flaming and smoldering, resulting in the generation of 12 different smoke samples. These data have been previously published in the following example environmental health research studies, with data made publicly available:

- Rager JE, Clark J, Eaves LA, Avula V, Niehoff NM, Kim YH, Jaspers I, Gilmour MI. Mixtures modeling identifies chemical inducers versus repressors of toxicity associated with wildfire smoke. Sci Total Environ. 2021 Jun 25;775:145759. doi: 10.1016/j.scitotenv.2021.145759. Epub 2021 Feb 10. PMID: 33611182 (https://pubmed.ncbi.nlm.nih.gov/33611182/).
- Kim YH, Warren SH, Krantz QT, King C, Jaskot R, Preston WT, George BJ, Hays MD, Landis MS, Higuchi M, DeMarini DM, Gilmour MI. Mutagenicity and Lung Toxicity of Smoldering vs. Flaming Emissions from Various Biomass Fuels: Implications for Health Effects from Wildland Fires. Environ Health Perspect. 2018 Jan 24;126(1):017011. doi: 10.1289/EHP2200. PMID: 29373863 (https://pubmed.ncbi.nlm.nih.gov/29373863/).

```
## Warning: package 'magrittr' was built under R version 4.1.2
```

Reading in file.

Probably many of you use `read.table` or `read.csv` to read in your file. This might not matter for many of you since reading in a small file does not take that long, but if you want to read a file that is in 100s of MBs of even multiple GBs, I suggest using `fread()` Why?




```
# Let's look at each columns and conduct basic analysis
# One of the way we can extract a data from a specific column is to use pull() function from tid
yverse
# Here we can see that there are multiple chemical types.
sample_data %>%
  pull(`Chemical Category`)
```

```
##  [1] "n-Alkanes"      "n-Alkanes"      "n-Alkanes"      "n-Alkanes"
##  [5] "n-Alkanes"      "n-Alkanes"      "n-Alkanes"      "n-Alkanes"
##  [9] "n-Alkanes"      "n-Alkanes"      "n-Alkanes"      "n-Alkanes"
## [13] "n-Alkanes"      "n-Alkanes"      "n-Alkanes"      "n-Alkanes"
## [17] "n-Alkanes"      "n-Alkanes"      "n-Alkanes"      "n-Alkanes"
## [21] "n-Alkanes"      "n-Alkanes"      "n-Alkanes"      "n-Alkanes"
## [25] "n-Alkanes"      "PAHs"           "PAHs"           "PAHs"
## [29] "PAHs"           "PAHs"           "PAHs"           "PAHs"
## [33] "PAHs"           "PAHs"           "PAHs"           "PAHs"
## [37] "PAHs"           "PAHs"           "PAHs"           "PAHs"
## [41] "PAHs"           "PAHs"           "PAHs"           "PAHs"
## [45] "PAHs"           "PAHs"           "PAHs"           "PAHs"
## [49] "PAHs"           "Methoxyphenols" "Methoxyphenols" "Methoxyphenols"
## [53] "Methoxyphenols" "Methoxyphenols" "Methoxyphenols" "Methoxyphenols"
## [57] "Methoxyphenols" "Methoxyphenols" "Methoxyphenols" "Methoxyphenols"
## [61] "Levoglucosan"   "Inorganics"     "Inorganics"     "Inorganics"
## [65] "Inorganics"     "Inorganics"     "Inorganics"     "Inorganics"
## [69] "Inorganics"     "Inorganics"     "Inorganics"     "Inorganics"
## [73] "Inorganics"     "Inorganics"     "Inorganics"     "Inorganics"
## [77] "Inorganics"     "Inorganics"     "Inorganics"     "Inorganics"
## [81] "Inorganics"     "Ions"           "Ions"           "Ions"
## [85] "Ions"           "Ions"
```

```
# You can also use unique() function afterwards to see unique categories
sample_data %>%
  pull(`Chemical Category`) %>%
  unique()
```

```
## [1] "n-Alkanes"      "PAHs"           "Methoxyphenols" "Levoglucosan"
## [5] "Inorganics"     "Ions"
```

```
# Using the table can tell you the # of each chemical type
sample_data %>%
  pull(`Chemical Category`) %>%
  table()
```

```
## .
##     Inorganics           Ions   Levoglucosan Methoxyphenols      n-Alkanes
##             20              5              1             11             25
##           PAHs
##             24
```

```
# Or... you can also do this!
# group_by() takes an existing data and converts it into a grouped data where operations are per
formed "by group"
sample_data %>%
  group_by(`Chemical Category`) %>%
  tally() %>%
  arrange(desc(n))
```

```
## # A tibble: 6 × 2
##   `Chemical Category`     n
##   <chr>               <int>
## 1 n-Alkanes              25
## 2 PAHs                   24
## 3 Inorganics             20
## 4 Methoxyphenols         11
## 5 Ions                    5
## 6 Levoglucosan            1
```

```
# In certain cases, you might need to subset your data by picking few of the columns.
# For this case, let's make a data subset.
# You can use select() function to select few columns.
# Also, you can use magrittr's set_colnames() function to rename the columns
data_subset <- sample_data %>%
  select(`Chemical Category`, Eucalyptus_Smoldering, Eucalyptus_Flaming) %>%
  set_colnames(c("category", "smoldering", "flaming"))

head(data_subset)
```

```
##     category smoldering flaming
## 1: n-Alkanes       0.06    0.06
## 2: n-Alkanes       0.04    0.04
## 3: n-Alkanes       0.21    0.25
## 4: n-Alkanes       0.04    0.04
## 5: n-Alkanes       0.11    0.25
## 6: n-Alkanes       0.13    0.28
```

```
# Also, you can subset this data further by using filter() function
data_subset %>%
  filter(flaming > 0.2) %>%
  head()
```

```
##     category smoldering flaming
## 1: n-Alkanes       0.21    0.25
## 2: n-Alkanes       0.11    0.25
## 3: n-Alkanes       0.13    0.28
## 4: n-Alkanes       0.06    0.22
## 5: n-Alkanes       0.74    0.26
## 6:      PAHs       0.80    0.68
```

```
# Or, we can make a new column by using a mutate() function.
data_subset <-
  data_subset %>%
  mutate(above_threshold = ifelse(smoldering > 0.2 & flaming > 0.2, TRUE, FALSE))

# Making a new subset data
data_above_threshold <-
  data_subset %>%
  filter(above_threshold == TRUE)

# and do some analysis
data_above_threshold %>%
  group_by(category) %>%
  summarize(
    min = min(smoldering),
    q1 = quantile(smoldering, 0.25),
    median = median(smoldering),
    mean = mean(smoldering),
    q3 = quantile(smoldering, 0.75),
    max = max(smoldering)
  )
```

```
## # A tibble: 5 × 7
##   category          min      q1  median    mean      q3    max
##   <chr>           <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1 Inorganics       0.26   0.712   8.93   35.7    28.9   287.
## 2 Levoglucosan   504.   504.    504.    504.    504.    504.
## 3 Methoxyphenols   0.35   4.72   17.7    37.7    37.3   166.
## 4 n-Alkanes        0.21   0.342   0.475   0.475   0.607   0.74
## 5 PAHs             0.24   0.38    0.52    0.52    0.66    0.8
```

```
data_above_threshold %>%
  group_by(category) %>%
  summarize(
    min = min(flaming),
    q1 = quantile(flaming, 0.25),
    median = median(flaming),
    mean = mean(flaming),
    q3 = quantile(flaming, 0.75),
    max = max(flaming)
  )
```

```
## # A tibble: 5 × 7
##   category          min      q1  median    mean      q3     max
##   <chr>           <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Inorganics       1.09   2.8    33.2   180.     97.3   1066.
## 2 Levoglucosan   264.   264.    264.    264.    264.     264.
## 3 Methoxyphenols   0.35   0.35    1.06    2.97    2.11    16.6
## 4 n-Alkanes        0.25   0.252   0.255   0.255   0.258    0.26
## 5 PAHs             0.68   1.03    1.38    1.38    1.74     2.09
```

```
# By comparing these two together, we can see which chemical category has both
# eucalpytus smoldering and flaming values that are larger than 0.2
sample_data %>%
  group_by(`Chemical Category`) %>%
  tally() %>%
  arrange(desc(n))
```

```
## # A tibble: 6 × 2
##   `Chemical Category`     n
##   <chr>               <int>
## 1 n-Alkanes              25
## 2 PAHs                   24
## 3 Inorganics             20
## 4 Methoxyphenols         11
## 5 Ions                    5
## 6 Levoglucosan            1
```

```
data_above_threshold %>%
  group_by(category) %>%
  tally() %>%
  arrange(desc(n))
```

```
## # A tibble: 5 × 2
##   category           n
##   <chr>          <int>
## 1 Inorganics        18
## 2 Methoxyphenols    11
## 3 n-Alkanes          2
## 4 PAHs               2
## 5 Levoglucosan       1
```

2. ggplot2

It is safe to assume that almost all of you know how to use ggplot. However, if you have never used ggplot, you can think of ggplot (https://ggplot2.tidyverse.org/) as a “prettier” version of base R plots.

```
library(MetBrewer)
# Let's melt the data so that we can visualize
# Melting simply makes the data longer take a look at what this function does.

# Previously...
head(sample_data)
```

```
##    Chemical Category           Chemical     CASRN Eucalyptus_Smoldering
## 1:         n-Alkanes 2-Methylnonadecane 1560-86-7                  0.06
## 2:         n-Alkanes 3-Methylnonadecane 6418-45-7                  0.04
## 3:         n-Alkanes           Docosane  629-97-0                  0.21
## 4:         n-Alkanes Dodecylcyclohexane 1795-17-1                  0.04
## 5:         n-Alkanes           Eicosane  112-95-8                  0.11
## 6:         n-Alkanes        Heneicosane  629-94-7                  0.13
##    Eucalyptus_Flaming Peat_Smoldering Peat_Flaming Pine_Smoldering Pine_Flaming
## 1:               0.06            1.36         0.06            0.06         0.06
## 2:               0.04            1.13         0.90            0.47         0.04
## 3:               0.25            9.46         0.57            0.16         0.48
## 4:               0.04            0.25         0.04            0.04         0.04
## 5:               0.25            7.55         0.54            0.17         0.29
## 6:               0.28            6.77         0.34            0.13         0.42
##    Pine_Needles_Smoldering Pine_Needles_Flaming Red_Oak_Smoldering
## 1:                    0.06                0.06               0.06
## 2:                    0.04                0.72               0.04
## 3:                    0.32                0.18               0.16
## 4:                    0.12                0.04               0.04
## 5:                    0.28                0.16               0.15
## 6:                    0.30                0.13               0.13
##    Red_Oak_Flaming     Units
## 1:            0.13 ng_per_uL
## 2:            0.77 ng_per_uL
## 3:            0.36 ng_per_uL
## 4:            0.04 ng_per_uL
## 5:            0.38 ng_per_uL
## 6:            0.69 ng_per_uL
```

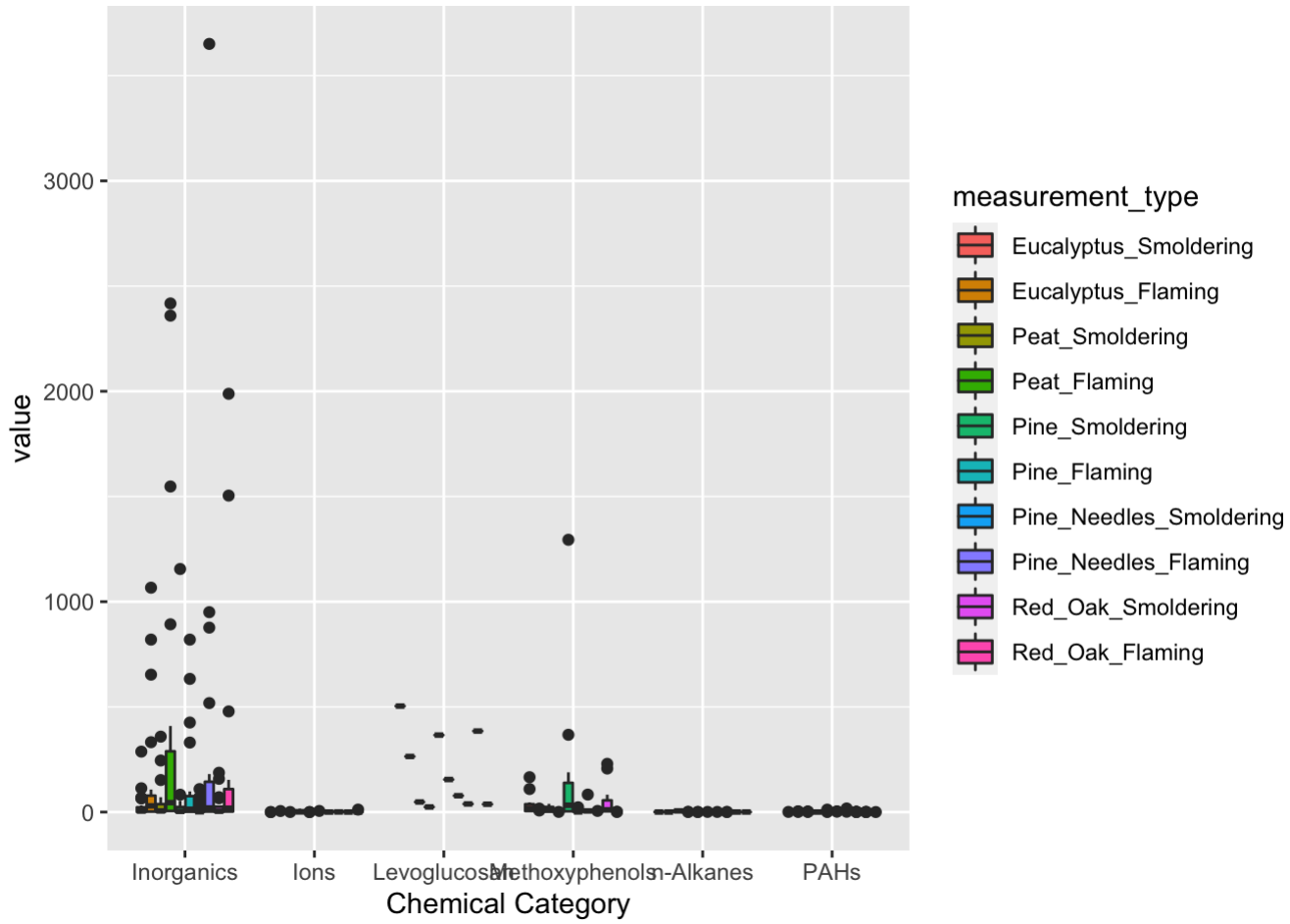
```
# With melt
sample_data_melt <- melt(sample_data, variable = "measurement_type")
```

```
## Using Chemical Category, Chemical, CASRN, Units as id variables
```

```
head(sample_data_melt)
```

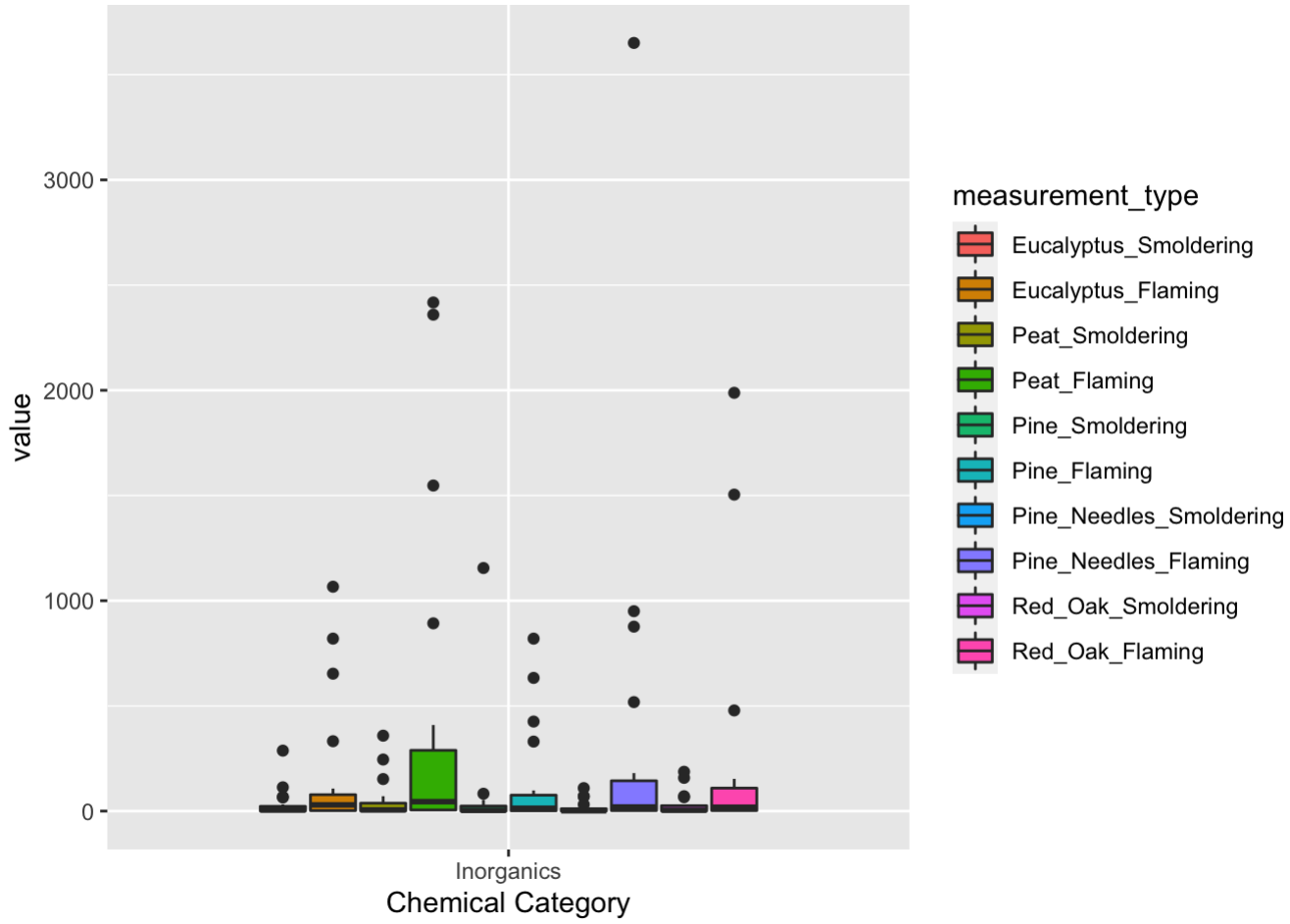
```
##   Chemical Category           Chemical     CASRN     Units
## 1         n-Alkanes 2-Methylnonadecane 1560-86-7 ng_per_uL
## 2         n-Alkanes 3-Methylnonadecane 6418-45-7 ng_per_uL
## 3         n-Alkanes           Docosane  629-97-0 ng_per_uL
## 4         n-Alkanes Dodecylcyclohexane 1795-17-1 ng_per_uL
## 5         n-Alkanes           Eicosane  112-95-8 ng_per_uL
## 6         n-Alkanes        Heneicosane  629-94-7 ng_per_uL
##        measurement_type value
## 1 Eucalyptus_Smoldering  0.06
## 2 Eucalyptus_Smoldering  0.04
## 3 Eucalyptus_Smoldering  0.21
## 4 Eucalyptus_Smoldering  0.04
## 5 Eucalyptus_Smoldering  0.11
## 6 Eucalyptus_Smoldering  0.13
```

```
ggplot(sample_data_melt, aes(x = `Chemical Category`, y = value, fill = measurement_type)) +
  geom_boxplot()
```

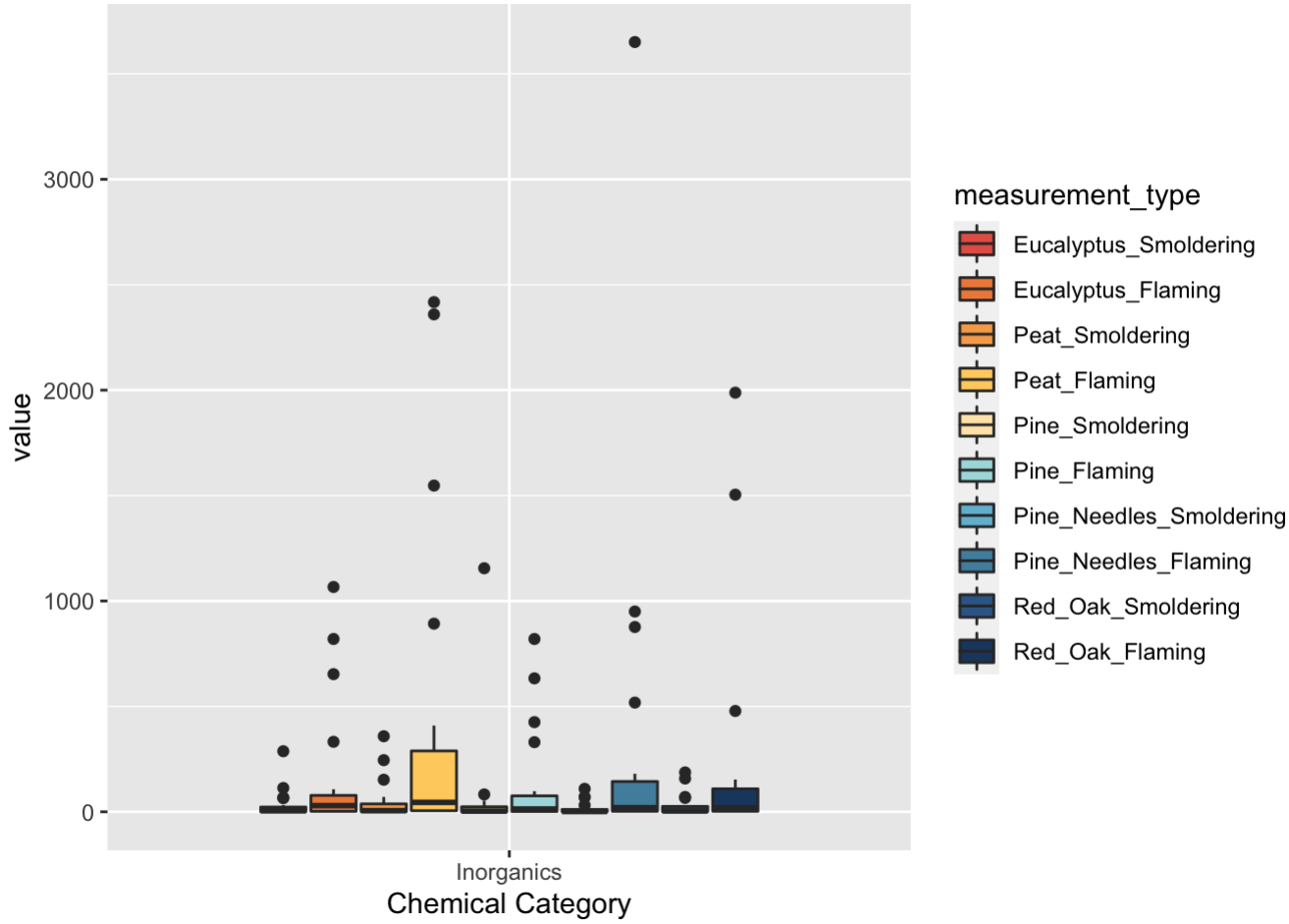


```
inorganic_data_melt <-
  sample_data_melt %>%
  filter(`Chemical Category` == "Inorganics")

ggplot(inorganic_data_melt, aes(x = `Chemical Category`, y = value, fill = measurement_type)) +
  geom_boxplot()
```




```
# Using "pretty" colors
ggplot(inorganic_data_melt, aes(x = `Chemical Category`, y = value, fill = measurement_type)) +
  geom_boxplot() +
  scale_fill_manual(values = met.brewer("Hiroshige", 10))
```



ggplot examples

Top 50 ggplot2 Visualizations (http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html)

R-graph gallery (https://www.r-graph-gallery.com/ggplot2-package.html)

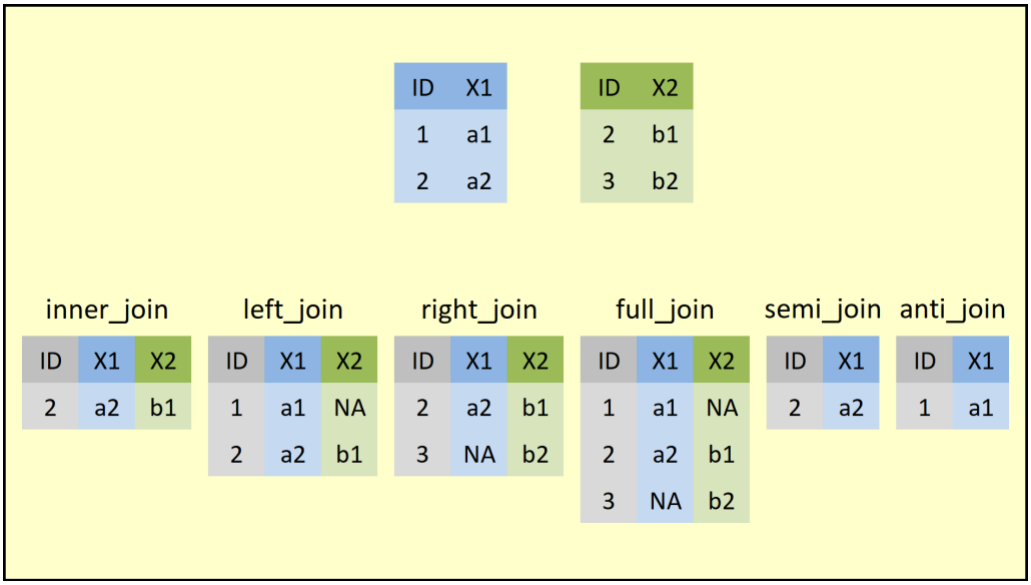
Other color palette options

MetBrewer (Current) (https://github.com/BlakeRMills/MetBrewer/tree/main)

WesAnderson (https://github.com/karthik/wesanderson)

PNWColors (https://github.com/jakelawlor/PNWColors)

Other useful tidyverse functions



Read more about it here: Joining Data Frames together (SQL Style) (https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti)

````
13
14 ```{python}
15 import pandas
16 flights = pandas.read_csv("flights.csv")
17 flights = flights[flights['dest'] == "ORD"]
18 flights = flights[['carrier', 'dep_delay', 'arr_delay']]
19 flights = flights.dropna()
20 ```
21
22 ```{r, fig.width=7, fig.height=3}
23 library(ggplot2)
24 ggplot(py$flights, aes(carrier, arr_delay)) + geom_point() + geom_jitter()
25 ```
26
````

Running python on R markdown: Reticulate (https://rstudio.github.io/reticulate/)

Reticulate Cheat Sheet (https://raw.githubusercontent.com/rstudio/cheatsheets/main/reticulate.pdf)